Przyczyny problemów

Jak wynika z przedstawionych wyżej badań i obserwacji, problemy nie dotyczą tylko pojedynczych projektów, ale całego przemysłu dostarczającego oprogramowanie. W 1967 roku można było ten stan — za konferencją NATO na temat inżynierii oprogramowania — nazwać kryzysem. Trwanie tego stanu do dzisiaj należy jednak, jak uznaje Roger Pressman, uznać za chorobę chroniczną. Można też połączyć te określe nia i nazwać je chronicznym kryzysem oprogramowania (Wyatt Gibbs, Software's Chronic Crisis, „Scientific American", 1994). O chroniczności tej sytuacji świadczy również to, że klasyczna książka Freda Brooksa (Legendarny osobomiesiąc, 1975) pozostaje nadal aktualna i wymagała jedynie krótkiego uwspółcześnienia w swym wydaniu jubileuszowym (z 1995 roku).

Jaka jest przyczyna tego „chronicznego kryzysu"? Na to pytanie w zasadzie odpo wiedź jest jedna: brak panowania nad wyjątkowo złożonym produktem, jakim jest system oprogramowania. Kluczowymi punktami są tutaj wymagania użytkowni ków, architektura i kontrola jakości. Wymagań użytkowników dla jednego systemu są zazwyczaj setki. Co więcej, są one sformułowane najczęściej przez odbiorców w taki sposób, że można na ich podstawie zbudować kilka całkowicie różnie dzia łających systemów i nadal będą one z nimi zgodne. Dlatego bardzo istotne jest wy konanie analizy wymagań precyzyjnie określającej sposób funkcjonowania systemu. Ten opis powinien być zrozumiały nie tylko dla użytkowników i analityków, ale również dla projektantów i programistów. Zadanie całkiem karkołomne, ale jeśli go nie wykonamy, to mamy problem! Mamy również problem, jeżeli nie założymy, że wymagania się zmieniają. Brak panowania nad zmieniającymi się wymaganiami to bardzo częsta przyczyna niepowodzeń.

Drugą przyczyną choroby oprogramowania jest architektura, a raczej jej brak. Architek tura jest takim elementem projektu, który zapewnia realizację wymagań użytkow nika, zapobiega „radosnej twórczości" programistów i znacznie redukuje złożoność tworzonych modeli. Jeżeli projektanci z dumą pokazują olbrzymie arkusze papieru, na których setki elementów i powiązań między nimi opisują strukturę naszego sys temu, to mamy problem. Wreszcie kontrola jakości. Często kojarzymy ją z testowa niem. Rzeczywiście — niedostateczne testowanie to częsty grzech firm produkują cych oprogramowanie. Równie poważnym grzechem jest jednak niedostateczna kontrola satysfakcji użytkowników. Jeżeli pokazujemy im działający system po raz pierwszy na kilka tygodni przed końcem projektu, to mamy problem.

Problemy, przed którymi stoi inżynieria oprogramowania, wynikają często z zanie chania stosowania ogólnych zasad inżynierii. Bardzo często zespoły tworzące oprogramowanie stosują raczej zasady pracy chałupniczej. Dlatego bardzo ważne jest, abyśmy zidentyfikowali przyczyny problemów i zaczęli je aktywnie pokonywać, stosując zasady wiedzy inżynierskiej z zakresu inżynierii oprogramowania.

Poniżej przedstawiono główne przyczyny występowania symptomów:

1. Nieprecyzyjne specyfikowanie oprogramowania. Język używany podczas formułowania wymagań często bardziej przypomina beletrystykę (np. powieści) niż precyzyjny opis techniczny. Przykład: „Czy »konto« oznacza »konto użytkownika«, czy »konto bankowe«?".
2. Zła komunikacja. Poszczególni uczestnicy projektu (analitycy wymagań, architekci, projektanci, użytkownicy) bardzo często rozmawiają ze sobą różnymi językami. Na przykład architekci używają diagramów niezrozumiałych dla analityków, a analitycy spisują wymagania, których nie są w stanie zrozumieć architekci (lub rozumieją je opacznie). Często też obieg informacji w projekcie jest tak długi, że praktycznie uniemożliwia wyjaśnianie wątpliwości. Przykład: „Och, to ten formularz jest taki istotny…".
3. Brak projektowania architektonicznego. Bardzo często pomija się całkowicie dyscyplinę projektowania. Przystępuje się do pisania kodu bezpośrednio po zebraniu wymagań. Wynika

to z przeświadczenia, że projektowanie spowalnia prace. Niestety, w przeważającej większości projektów (wyjątkiem są bardzo małe systemy) brak projektu prowadzi do chaosu i częstych nieporozumień między programistami. Zauważmy, że w budownictwie nikt przy zdrowych zmysłach nie zbuduje domu bez projektu (chociaż można tak zbudować budę dla psa). Przynosi to bardzo opłakane skutki. Przykład (na tydzień przed oddaniem systemu): „To właściwie jak mam się połączyć z komponentem obsługi płatności?".

4. Brak zarządzania złożonością systemu. Przyczyna ta wiąże się z projektowaniem systemu. Zarządzanie złożonością polega na umiejętnym podziale systemu na mniejsze fragmenty (moduły, pakiety, komponenty). Bez takiego podziału przestajemy panować nad systemem — przestaje się on „mieścić w głowie". Oznacza to, że każda zmiana, poprawka czy uzupełnienie okupywane są bardzo żmudnym i kosztownym odkrywaniem „jak to właściwie działa" czy też „co tak właściwie mieliśmy na myśli". Przykład: „Nasza baza danych ma 1500 tabel i 2000 klas dostępowych, to wszystko w jednym pakiecie, a mimo to jakoś sobie dajemy radę".
5. Bardzo późne odkrywanie poważnych nieporozumień. Złożoność problemu i budowanego systemu powoduje, że często dochodzi do nieporozumień między uczestnikami projektu. Odkrycie takiego nieporozumienia późno w procesie wytwarzania systemu może oznaczać dramatyczny wręcz wzrost kosztów. Szczególnie nieporozumienia dotyczące wymagań są opłakane w skutkach i mogą doprowadzić do konieczności rozpoczęcia budowy systemu praktycznie od początku. Przykład: „I dopiero teraz mi mówisz, że ta procedura nie ma parametru X?".
6. Brak zarządzania zmianami. Dla większości projektów produkcji oprogramowania można sformułować jeden pewnik: zmiany będą występować. Zmiany te są spowodowane różnymi czynnikami, ale podstawowym ich źródłem są zmieniające się wymagania. Zmianom tym nie można zapobiec (wynikają często z przyczyn obiektywnych), jednak brak zarządzania zmianami prowadzi do chaosu. Bez kontroli zmian nie sposób ocenić, jak duży będzie system po dokonaniu zmian. Prowadzi to do „puchnięcia systemu", czyli znacznego przekroczenia zakładanego zakresu prac.
7. Nieużywanie narzędzi wspomagających. Bez narzędzi stoimy w sytuacji architekta, który projekt budynku musi narysować za pomocą rysika i ekierki. Jest to możliwe do wykonania, ale bardzo nieefektywne w erze narzędzi CAD. Mimo to typową postawą zespołu wytwórczego jest: „Wszystko, czego nam trzeba, to dobry kompilator".

Dobrą metodą oceny aktualnego stanu organizacji wytwarzającej oprogramowanie jest przeprowadzenie tzw. badania dojrzałości do produkcji oprogramowania. Zasady takiego badania zostały stworzone już w latach 80. XX wieku przez Software Engi neering Institute Uniwersytetu Carnegie Mellon (www.sei.cmu.edu) pod nazwą Capability Maturity Model for Software (SW-CMM).

Model CMM posiada pięć poziomów. Każdy z nich określa coraz wyższą dojrzałość organizacji. Przyporządkowanie do któregoś z poziomów stanowi dobrą diagnozę jej stanu. Pierwszy poziom to poziom początkowy (ang. initial). Jego cechą charak terystyczną jest brak procesu; działania podejmowane są ad hoc, często w sposób chaotyczny. Na poziomie drugim — powtarzalnym (ang. repeatable) — istnieją już (choć często nieformalnie) i są przestrzegane podstawowe standardy prowadzenia projektów. Poziom trzeci zakłada istnienie zatwierdzonego standardu procesu wytwór czego, który jest przestrzegany we wszystkich projektach. Jest to zatem poziom zdefiniowany (ang. defined). Poziomy czwarty (zarządzany, ang. managed) i piąty (optymalizujący, ang. optimizing) oznaczają coraz bardziej szczegółowe przestrze ganie najlepszych praktyk inżynierii oprogramowania.

Najlepsze praktyki inżynierii oprogramowania

Coraz więcej organizacji wytwarzających oprogramowanie (chociaż nadal zdecy dowanie zbyt mało) zaczyna stosować metody inżynierskie w produkcji oprogra mowania. Jakie są zatem te metody? Jakie praktyki należy stosować, aby nie doświad czać symptomów „przewlekłej choroby"? Przedstawiamy tutaj zbiór najlepszych praktyk (ang. best practices) — sprawdzonych zasad związanych z tworzeniem oprogramowania. Zasady te, zastosowane w połączeniu, uderzają w podstawowe przyczyny problemów i niepowodzeń procesu. Nazwa „najlepsze praktyki" wynika nie z faktu, że możemy zmierzyć ich poziom „dobroci". Wynika raczej ze stwierdze nia, że są one najpowszechniej używane w organizacjach, które odnoszą sukcesy w produkcji oprogramowania. Najlepsze praktyki zostały zaczerpnięte od tysięcy takich organizacji i były stosowane w tysiącach projektów. Oto one:

- Produkuj iteracyjnie, czyli oddawaj system w sposób przyrostowy. Kolejne przyrosty zapobiegną jednemu „wielkiemu wybuchowi" pod koniec projektu.
- Stosuj architektury komponentowe, czyli panuj nad złożonością systemu, miej spójną koncepcję całości i określ ramy dla radosnej twórczości projektan tów i programistów. Panowanie nad złożonością systemu ułatwiają kompo nenty realizujące zasadę ukrywania informacji. Stosowanie komponentów promuje też zasadę ponownego wykorzystania (ang. reuse). To znamienne, że na tej samej konferencji NATO, na której ogłoszono kryzys oprogramowania, M.D. McIlroy opisał ideę komponentów, która dopiero obecnie może stać się panaceum na ten kryzys.
- Stale kontroluj jakość, czyli sprawdzaj poprawność systemu od samego po czątku projektu. Kontrola jakości nie powinna się ograniczać do testowania. Powinna polegać na ciągłym sprawdzaniu zgodności systemu z wymaganiami i poziomu satysfakcji odbiorców.
- Zarządzaj wymaganiami, czyli traktuj wymagania jak jednostki, które podle gają procedurom zarządzania. Każde z wymagań powinno posiadać ślad łą czący je z wymaganiami szczegółowymi, projektem interfejsu użytkownika oraz komponentami, które je realizują.
- Zarządzaj zmianami, czyli z góry załóż, że zmiany na pewno będą ciągle zgła szane. Często przekłada się to na istnienie oficjalnego „organu zgłaszania zmian".
- Modeluj wizualnie, czyli twórz diagramy opisujące różne aspekty systemu, gdyż dobry rysunek jest wart więcej niż tysiąc słów. Językiem budowy systemu powinien być język graficzny. Język powinien być wspólny i zrozumiały dla wszystkich, włącznie z przyszłymi użytkownikami.

Zaaplikowanie wszystkich tych praktyk pozwoli nam znaleźć się co najmniej na drugim lub trzecim poziomie CMM. Ważne jest jednak, aby nie aplikować ich wszystkich naraz! Doświadczony lekarz przeprowadza kurację stopniowo. Dlatego tak ważna jest ocena stanu organizacji. Dopiero na tej podstawie możemy opracować plan działania. Warto zauważyć, że poszczególne praktyki wzajemnie się uzupełniają. Trudno na przykład wyobrazić sobie ciągłą kontrolę jakości bez cyklu iteracyjnego.

Powyższe praktyki koncentrują się przede wszystkim na procesie technicznym i odpowiednim ujęciu go w formalne ramy konkretnych działań w różnych dyscyplinach. Inne podejście do praktyki inżynierii oprogramowania zostało wyrażone w tzw. manifeście zwinnego (ang. agile) wytwarzania oprogramowania (agilemanifesto.org). Brzmi on następująco:

„Odkrywamy nowe metody programowania dzięki praktyce w programowaniu i wspieraniu w nim innych. W wyniku naszej pracy zaczęliśmy bardziej cenić:

- *ludzi i interakcje od procesów i narzędzi,*
- *działające oprogramowanie od szczegółowej dokumentacji,*
- *współpracę z klientem od negocjacji umów,*

- *reagowanie na zmiany od realizacji założonego planu.*

Oznacza to, że elementy wypisane po prawej są wartościowe, ale większą wartość mają dla nas te, które wypisano po lewej".

Przykładowe dziedziny zastosowań inżynierii oprogramowania

Bankowość elektroniczna

- Otwieranie rachunków i dokonywanie transakcji.
- Obsługa kredytów i kart kredytowych.
- Proces przyznawania kredytu (od złożenia wniosku o kredyt do przesłania informacji o decyzji i uruchomienia kredytu).

Biblioteka

- Obsługa księgozbioru oraz wypożyczeń.
- Rejestrowanie czytelników oraz rozliczenia z czytelnikami.
- Proces wypożyczenia książki (od złożenia zamówienia do wydania książki).

Biuro geodezyjne

- Obsługa prac geodezyjnych zlecanych przez różne podmioty.
- Rejestrowanie prac geodezyjnych w systemach informacji geodezyjnej.
- Proces wykonania zlecenia wytyczenia obszaru w terenie (od otrzymania zlecenia do zarejestrowania wyniku wytyczenia).

Biuro podróży

- Obsługa definiowania wycieczek i kontaktów z dostarczycielami usług tury stycznych (hotelami, przewoźnikami).
- Obsługa sprzedaży i zakupu wycieczek.
- Proces uczestnictwa w wycieczce (od złożenia zamówienia do powrotu z wycieczki).

Ewidencja pojazdów i kierowców

- Zarządzanie wydziałami komunikacji (struktura organizacyjna, godziny pracy itp.).
- Rejestracja i udostępnianie danych o prawach jazdy.
- Rejestracja pojazdów.
- Proces rejestracji pojazdów (od złożenia wniosku do otrzymania dowodu rejestracyjnego).